

Exceptions

CSCI 334
Stephen Freund

Fortran Control Structure

```
10 IF (X > 0.0) GO TO 20
11 X = -X
   IF (X < 0.0) GO TO 50
20 IF (X * Y < 0.0) GO TO 30
   X = X - Y - Y
30 X = X + Y
   ...
50 ...
   X = A
   Y = B - A
   GO TO 11
   ...
```



Block Structured Programming

```
if (x < 0) {
    x = -x;
    while (y > 0) {
        y = b - a;
        x = x / z;
        if (x > 10) {
            x = x / 2;
        } else {
            x = x * 2;
        }
    }
}
```

COM Example

```
/* Call A Remote Method on a COM object. Returns 0 on
success, 1 on failure. */
int callRemoteMethod() {
    int hResult;
    IDispatch pIDispatch;
    hResult=CLSIDFromProgID(...);
    if (FAILED(hResult)) return 1;
    hResult=CoInitialize(...);
    if (FAILED(hResult)) return 1;
    hResult=CoCreateInstance(pIDispatch,...);
    if (FAILED(hResult)) return 1;
    hResult=punk->QueryInterface(...);
    if (FAILED(hResult)) { pIDispatch->Release(); return 1; }
    hResult = pIDispatch->GetIDsOfNames(...);
    if (FAILED(hResult)) { pIDispatch->Release(); return 1; }
    hResult = pIDispatch->Invoke(...);
    if (FAILED(hResult)) { pIDispatch->Release(); return 1; }
    return 0;
}
```

Exceptions Preview

```
void callRemoteMethod() {
    IDispatch pIDispatch;
    try {
        CLSIDFromProgID(...);
        CoInitialize(...);
        CoCreateInstance(pIDispatch);
        punk->QueryInterface(...);
        pIDispatch->GetIDsOfNames(...);
        pIDispatch->Invoke(...);
    } catch (Exception e) {
        if (pIDispatch != null) pIDispatch->Release();
        throw e;
    }
}
```

```
void CoInitialize() {
    ...
    if (bad)
        throw new Exception();
    ...
}
```

Stack Example

```
type Stack = int list;
```

```
fun evalX(nil,a::st) = a
| evalX(PUSH(n)::rest,st) = evalX(rest, n::st)
| evalX(ADD::rest, a::b::st) = evalX(rest, (b+a)::st)
| evalX(MULT::rest, a::b::st) = evalX(rest, (b*a)::st)
| evalX(DIV::rest, a::b::st) = evalX(rest, (b div a)::st)
| evalX(SUB::rest, a::b::st) = evalX(rest, (b-a)::st)
| evalX(SWAP::rest, a::b::st) = evalX(rest, b::a::st)
| evalX(_,_) = 0
;
```

```
fun eval(instrs, stack) =
    print Int.toString(evalX(instrs, stack));
```

Stack Example

```
type Stack = int list;

exception DivideByZero;
exception BadOp;

fun evalX (nil,a::st) = a
| evalX (PUSH(n)::rest,st) = evalX (rest, n::st)
| evalX (ADD::rest, a::b::st) = evalX (rest, (b+a)::st)
| evalX (DIV::rest, 0::b::st) = raise DivideByZero
| evalX (DIV::rest, a::b::st) =
    evalX (rest, (b div a)::st)
...
| evalX (_,_) = raise BadOp;
```

Stack Example

```
fun eval(instrs, stack) =
    (
        print Int.toString(evalX(instrs, stack))
    ) handle BadOp => print "Bad Operation"
        | DivideByZero => print "Div by 0";

- eval([PUSH(3), PUSH(0), DIV], nil);
Div by 0
```

Stack Example (2)

```
type Stack = int list;

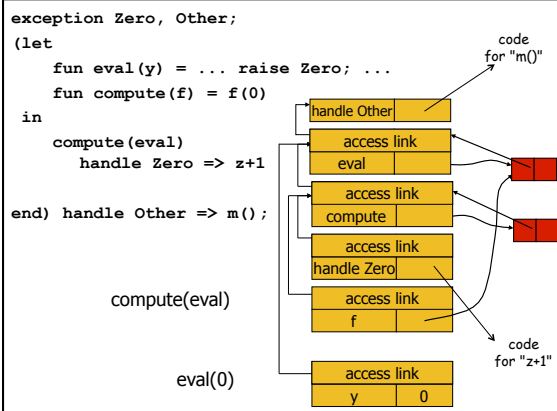
exception EvalError of string;

fun evalX (nil,a::st) = a
| evalX (PUSH(n)::rest,st) = evalX (rest, n::st)
| evalX (ADD::rest, a::b::st) = evalX (rest, (b+a)::st)
| evalX (DIV::rest, 0::b::st) =
    raise EvalError("Div by 0")
| evalX (DIV::rest, a::b::st) =
    evalX (rest, (b div a)::st)
...
| evalX (_,_) =
    raise EvalError("Bad op");
```

Stack Example (2)

```
fun eval(instrs, stack) =
    (
        print Int.toString(evalX(instrs, stack))
    ) handle EvalError(msg) => print msg;

- eval([PUSH(3), PUSH(0), DIV], nil);
Div by 0
```



Checked Exceptions in Java

- class IOException extends Exception { ... }
- class FileReader {
 public FileReader(String name) ... {
 if (error happens) throw new IOException();
 }
 }
- try {
 FileReader in = new FileReader("input.txt");
 ..
 } catch (IOException e) {
 ...
 } catch (NetworkFailureException e) {
 ...
 }
 }

Declared Exceptions

```
public FileReader(String name) throws IOException {
    if (error happens) throw IOException();
}

void m1() {
    try {
        ... new FileReader(...);
    } catch (IOException e) { ... } ← function handles exn.
}

void m2() throws IOException { ← caller must handle exn.
    ... new FileReader(...);
}

void m3() { m2(); } ← BAD
```

Resource Management (Memory)

```
try {
    ...
    // creates lots o' memory
    ...
} catch (Exception e) {
    ...
}
```

- GC works well with exceptions to clean up mem.
- C++ or no GC:
 - much harder
 - where do you free mem?

Resource Management (Other...)

- Files, sockets, DB connections
- Limited number available
 - must be released when done
- Without Exceptions:

```
Socket f;
...
f = new Socket("bull.cs.williams.edu", 80);
...
f.read();
...
f.close();
```

Resource Management (Other...)

- Files, sockets, DB connections
- Limited number available
 - must be released when done
- With Exceptions:

```
Socket f =
    new Socket(...);

try {
    ...
    f.read();
} catch (SocketException e) {
}
```

When is f closed?
· on exit from try-block
· on exit from handler
· on exit caused by
uncaught exception

Finalizers

- GC calls finalize on objects right before collection:

```
class Socket {
    private OSSocket osHandle;

    void close() { release(osHandle); }

    void finalize() {
        this.close();
    }
}
```

- Problems?

Try-Finally Blocks

```
Socket f = new Socket(...);
```

```
try {
    ...
} catch (Exception e) {
    ...
} finally {
    f.close();
}
```

- finally code runs:
 - when control leaves try part (normally or exceptionally)
 - when control leaves exception handler